Chapitre I

ETAT DE L'ART DES ALGORITHMES D'OPTIMISATION

I.1. INTRODUCTION

L'optimisation de forme a fait l'objet de nombreux travaux [*Kusiak et al. 1989*], [*Balan 1996*], [*Vieilledent 1999*], [*Castro el al. 2000*], [*Antonio al. 2002*], etc. Grâce à ces études, la performance et l'efficacité des algorithmes d'optimisation ont beaucoup évoluées. Il est nécessaire de faire une présentation de ces algorithmes avant de présenter la nouvelle approche d'hybridation. Ce chapitre a cet objectif.

Un problème d'optimisation de forme peut généralement être présenté comme suit:

$$\begin{cases} Minimiser\, \Phi(\mu) \\ c_i(\mu) \le 0 \quad \forall i = 1,...,m_i \\ h_i(\mu) = 0 \quad \forall i = 1,...,m_e \\ \mu \in \Re^n \end{cases} \tag{1.1}$$

$\Phi(\mu)$ représente la fonction coût, qui peut être une combinaison de plusieurs critères ; $c_i(\mu)$ et $h_i(\mu)$ désignent les contraintes d'inégalité et d'égalité auxquelles est soumis le vecteur des paramètres à optimiser μ.

Selon les circonstances (résultats expérimentaux, disponibilité du gradient de $\Phi(\mu)$, etc.) ainsi que les ressources disponibles (nombre possible d'évaluations, capacité des ordinateurs, etc.), le problème d'optimisation (1.1) pourra être résolu par différents algorithmes. Citons certains d'entre eux comme l'algorithme de gradient conjugué, les algorithmes de Newton, les algorithmes génétiques, les stratégies d'évolution, les méthodes de surface de réponse, etc. Nous pouvons classifier ces algorithmes d'optimisation en 3 catégories comme suit :

- Algorithmes à direction de descente (algorithmes à base de gradient)
- Algorithmes d'ordre 0 et algorithmes évolutionnaires
- Algorithmes hybrides

Nous allons maintenant présenter ces trois classes d'algorithmes d'optimisation.

I.2. METHODES A DIRECTION DE DESCENTE

Dans cette section, nous introduisons une classe importante d'algorithmes de résolution des problèmes d'optimisation : les algorithmes à direction de descente. Ce sont des algorithmes qui ont obligatoirement besoin de l'information du gradient des fonctions coût pour chercher

l'optimum du problème. Leur utilisation nécessite que la fonction coût soit au moins une fois différentiable par rapport aux paramètres à optimiser. Nous en décrirons, dans la première partie de cette section, les principes généraux.

I.2.1. Principes généraux

Pour utiliser avec cette catégorie d'algorithmes, supposons que la fonction Φ est continue et différentiable dans tout l'espace de recherche. Nous notons $\nabla\Phi(\mu)$ et $\nabla^2\Phi(\mu)$ respectivement le vecteur gradient et la matrice hessienne de la fonction coût $\Phi(\mu)$ en μ. La condition nécessaire d'optimalité du problème d'optimisation (1.1) s'écrit :

$$\nabla\Phi(\mu) = 0 \tag{1.2}$$

Lorsque la fonction coût $\Phi(\mu)$ est deux fois différentiable, on peut écrire une condition suffisante d'optimalité :

$$\begin{cases} \nabla\Phi(\mu) = 0 \\ \nabla^2\Phi(\mu) \quad \text{définie positive} \end{cases} \tag{1.3}$$

Les méthodes à direction de descente ont pour objectif de calculer un vecteur μ satisfaisant la condition nécessaire d'optimalité (1.2).

Pour une fonction coût donnée $\Phi(\mu)$, on dit que d est une direction de descente de $\Phi(\mu)$ en $\mu \in \Re^n$ si la relation suivante est vérifiée :

$$d.\nabla\Phi(\mu) < 0 \tag{1.4}$$

Géométriquement cela veux dire que d fait avec l'opposé du gradient $-\nabla\Phi(\mu)$ un angle θ strictement plus petit que $\frac{\pi}{2}$.

Si d est une direction de descente, pour tout $\lambda > 0$ suffisamment petit, en utilisant un développement de Taylor d'ordre 1 nous avons :

$$\Phi(\mu + \lambda d) < \Phi(\mu) \tag{1.5}$$

Nous voyons que $\Phi(\mu)$ décroît dans la direction d. Les directions de descentes sont intéressantes car, pour faire diminuer $\Phi(\mu)$, il suffit de faire un déplacement de μ le long de d. L'algorithme général construit une suite d'itérés μ^k approchant une solution μ^* du problème (1.1) par la récurrence :

$$\begin{cases} \mu^0 \in \Re^n \\ \mu^{k+1} = \mu^k + \lambda^k d^k \quad , \forall k \geq 0 \end{cases} \tag{1.6}$$

où $\lambda^k > 0$ est appelé le pas de descente, d^k une direction de descente de Φ en μ^k.

Le pas de descente doit être déterminé de telle sorte que la valeur de la fonction décroisse le plus possible. En général, un algorithme de recherche linéaire est utilisé pour trouver un pas "optimal" à chaque itération.

L'algorithme général de cette classe de méthodes d'optimisation pour résoudre un problème sans contrainte s'écrit :

Choix d'un itéré initial $\mu^0 \in \Re^n$, et d'un paramètre d'arrêt ε

Pour $k \geq 0$

1. *tant que $\left\|\nabla\Phi\left(\mu^k\right)\right\| > \varepsilon$, continue ;*
2. *recherche d'une direction de descente d^k ;*
3. *recherche linéaire : déterminer un pas $\lambda^k > 0$ minimisant $\Phi\left(\mu^k + \lambda d^k\right)$*
4. *actualisation pour l'itération suivante : $\mu^{k+1} = \mu^k + \lambda^k d^k$; $k = k+ 1$*
5. *aller à l'étape 1.*

Le test d'arrêt apporté à l'étape 1 porte sur la condition nécessaire d'optimalité d'ordre 1 : $\left\|\nabla\Phi\left(\mu^k\right)\right\| \approx 0$.

Pour définir une méthode de direction de descente, il faut donc préciser deux ingrédients principaux qui sont :

- la manière de choisir la direction de descente, qui donne le nom de l'algorithme
- la méthode de recherche linéaire pour déterminer le pas de descente λ optimal à chaque itération

Quelques méthodes correspondant à des choix particuliers ces deux ingrédients sont décrits dans les paragraphes qui suivent.

I.2.2. Choix de la direction de descente

I.2.2.1. Méthode de la plus forte pente

Il s'agit de la méthode à direction de descente d'ordre 1 la plus simple. L'inverse du gradient est évidemment une direction de descente si μ^k n'est pas un point stationnaire. La direction normalisée *d* de plus forte décroissance de Φ au voisinage d'un point μ est donnée par :

$$d = -\frac{\nabla\Phi(\mu)}{\left\|\nabla\Phi(\mu)\right\|} \tag{1.7}$$

La méthode du gradient est facile à mettre en œuvre. Cependant, sa convergence peut être très lente. Loin de la solution, cette direction est une bonne direction de descente. En revanche, dans le voisinage d'une solution optimale μ^* du problème, là où les termes du second ordre de $\Phi(\mu)$ en μ^* jouent un rôle plus important, on observe une diminution très lente de Φ. Cela est son défaut majeur. Il existe des techniques d'accélération permettant de palier à cet inconvénient [*Minoux 1983*]. Néanmoins, elles sont fort coûteuses et peu utilisées. Dans le domaine de mise en forme, *Jensen et al.* [*Jensen et al. 1998*] se sont servis cette méthode pour la minimisation de l'usure en emboutissage.

I.2.2.2. Méthode du gradient conjugué

L'algorithme du gradient conjugué [*Morris 1982*] peut être vu comme une légère modification de l'algorithme de la plus forte pente, pour lequel on garde en mémoire la direction de descente de l'itération précédente. La direction de descente est donnée par :

$$d^k = \begin{cases} -\nabla\Phi\left(\mu^1\right) & si \quad k=1 \\ -\nabla\Phi\left(\mu^k\right)+\beta^{k-1}d^{k-1} & si \quad k\geq 2 \end{cases} \tag{1.8}$$

Le scalaire β^k peut prendre différentes valeurs, ce qui donne à l'algorithme des propriétés différentes. Nous présentons ici les deux méthodes qui sont très fréquemment citées dans la littérature. Elles calculent β^k par les formules suivantes :

- Méthode de Fletcher-Reeves [*Fletcher et al. 1964*] :

$$\beta^k = \frac{\left\|\nabla\Phi\left(\mu^k\right)\right\|^2}{\left\|\nabla\Phi\left(\mu^{k-1}\right)\right\|^2} \tag{1.9}$$

- Méthode de Polak-Ribière [*Polak et al. 1969*] :

$$\beta^k = \frac{\left[\nabla\Phi\left(\mu^k\right)-\nabla\Phi\left(\mu^{k-1}\right)\right]^T\nabla\Phi\left(\mu^k\right)}{\left\|\nabla\Phi\left(\mu^{k-1}\right)\right\|^2} \tag{1.10}$$

Cet algorithmes est proposé à l'origine pour la minimisation de fonctions quadratiques, puis il est étendu à des fonctions quelconques dans [*Fletcher et al. 1964*]. Néanmoins, il cumule les erreurs d'arrondi, la convergence n'est alors assurée que si l'on procède à des réinitialisations périodiques. Certaines techniques de redémarrage ont été proposées (méthode de Beales [*Powell 1975*] par exemple). *Zabaras et al.* [*Zabaras et al. 1995*] ont utilisé cette méthode pour l'optimisation du flux de chaleur en solidification.

I.2.2.3. Méthode de Newton

L'algorithme général de Newton est une méthode de résolution de systèmes d'équations non linéaires. Dans le cadre de l'optimisation, il est utilisé comme un algorithme de direction de descente sur la condition nécessaire d'optimalité (1.2).

Pour décrire cet algorithme, nous considérons l'approximation quadratique $Q(\mu)$ de $\Phi(\mu)$ au voisinage d'un point μ^k à l'itération *k*:

$$Q(\mu) = \Phi(\mu^k) + \nabla\Phi(\mu^k).(\mu - \mu^k) + \frac{1}{2}(\mu - \mu^k)^t.\nabla^2\Phi(\mu^k).(\mu - \mu^k) \tag{1.11}$$

On choisit alors μ^{k+1} qui minimise Q. Il est définit par la condition d'optimalité (1.2) ce qui conduit au système linéaire suivant pour déterminer la direction de descente d^k:

$$d^k = -\left[\nabla^2\Phi(\mu^k)\right]^{-1}.\nabla\Phi(\mu^k) \tag{1.12}$$

Cette méthode est reconnue comme une des plus efficaces dans la famille des algorithmes à direction de descente. Sa principale difficulté réside dans le calcul des dérivées secondes de Φ qui s'avère le plus souvent coûteux et difficile à réaliser.

Plusieurs algorithmes proposent de lever cette difficulté en utilisant une approximation de la matrice hessienne. On peut mentionner le cas particulier où Φ peut s'écrire sous forme de moindres carrés. On obtient alors une approximation du hessien en ne considérant que les produits des gradients. Cette approximation, qui est à la base des algorithmes de Gauss-Newton ou Levenberg-Marquardt [*Minkowycz 1988*], est largement utilisée en identification de paramètres rhéologiques [*Gavrus 1996*] [*Ghouati 1998*].

I.2.2.4. Méthodes quasi-Newton

L'idée des méthodes de quasi-Newton est de remplacer la matrice hessienne $\nabla^2\Phi(\mu^k)$ (ou son inverse) par une approximation mise à jour itérativement. La direction de descente s'écrit :

$$d^k = -\left[\tilde{H}^k\right]^{-1}.\nabla\Phi(\mu^k) \tag{1.13}$$

où la matrice $\tilde{H}^k$ doit converger vers la matrice hessienne lorsqu'on s'approche de l'optimum. Pour tout *k*, la matrice $\tilde{H}^k$ doit vérifier plusieurs conditions. D'une part, elle doit être symétrique et définie positive. D'autre part, on impose à la matrice $\tilde{H}^{k+1}$ de vérifier la relation de quasi-Newton suivante :

$$\nabla\Phi(\mu^{k+1}) - \nabla\Phi(\mu^k) = \tilde{H}^{k+1}(\mu^{k+1} - \mu^k) \tag{1.14}$$

Plusieurs méthodes pour construire la suite $\tilde{H}^k$ ont été proposées. Les deux méthodes les plus connues sont de rang 2, celle DFP de Davidon-Fletcher-Powell [*Fletcher et al. 1963*] et celle BFGS [*Broyden et al. 1970*].

Pour un problème d'extrusion, Kusiak et al. [*Kusiak et al. 1989*] ont montré l'efficacité des méthodes de quasi-Newton en comparant une méthode DFP, un algorithme de plus forte pente et une méthode d'ordre 0 le simplex. Avec de nombreuses utilisations comme dans [*Noiret et al. 1996*], [*Zhao et al. 1997*], [*Bourdin et al. 1998*], [*Vieilledent et al. 1998*], la méthode BFGS semble la plus adaptée aux problèmes d'optimisation en mise en forme.

I.2.3. Méthodes de recherche linéaire

Une fois que la direction de descente d^k à l'itération k est déterminée, on effectue l'étape de recherche linéaire pour trouver la valeur optimale du pas de descente λ^k. La recherche linéaire permet aux méthodes à direction de descente de converger, la convergence n'étant pas assurée si l'on néglige cette étape. Cette dernière constitue souvent la partie la plus coûteuse en temps de calcul, car elle nécessite plusieurs évaluations de la fonction Φ. Le pas de descente λ^k trouvé doit satisfaire deux conditions : Faire décroître Φ et ne pas être trop petit pour éviter une fausse convergence.

La recherche linéaire est un problème d'optimisation avec un seul paramètre. Il existe donc plusieurs méthodes et plusieurs critères d'arrêt de cette recherche tels que celui de Curry, celui d'Admijo ou Goldstein [*Armijo 1966*], ou celui de Wolf [*Wolf 1971*] pour BFGS. Nous pouvons citer ici quelques méthodes de recherche linéaire : celle de Newton (ou de la sécante), la méthode d'interpolation quadratique [*Fletcher 1996*], la méthode de la section d'or [*Morris 1982*] [*Chung et al. 1998*], la méthode de Brent avec gradient [*Chen et al. 1995*], la méthode d'interpolation parabolique [*Press et al. 1992*], la méthode de Davidon [*Schittkowski 1980*], la méthode de type Moré et Thuente [*Moré et al. 1994*], etc.

I.2.4. Avantages – Inconvénients

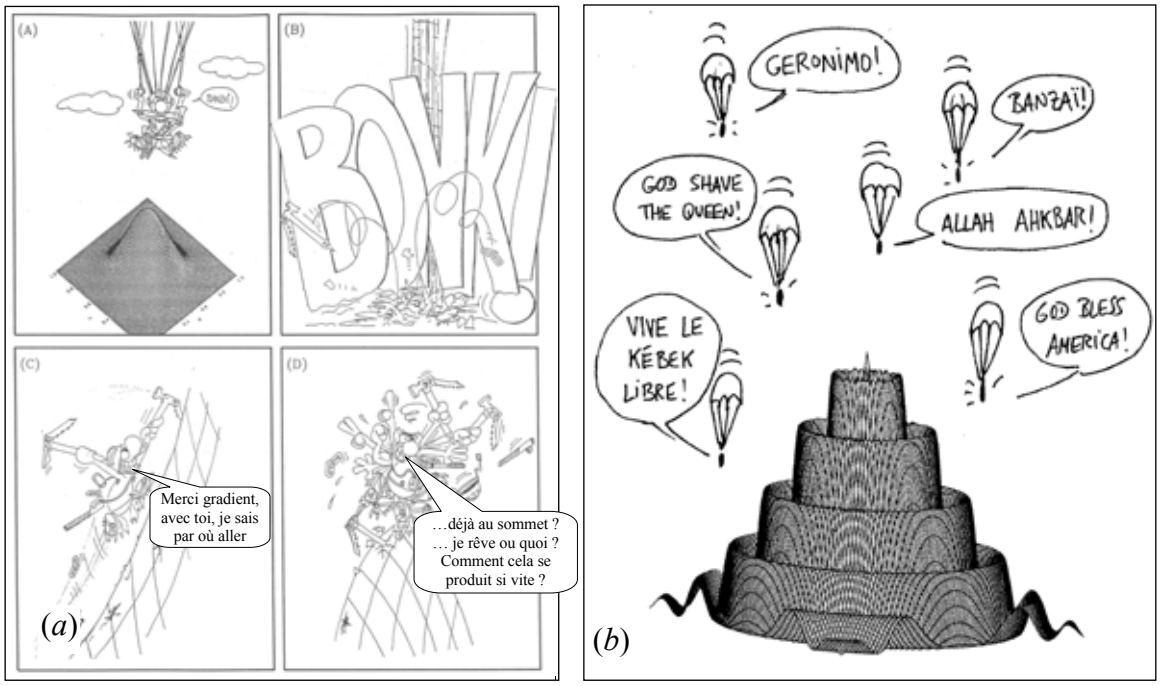

Figure 1.1. Méthode à direction de descente: devant un problème convexe, je trouve rapidement l'optimum (a); devant un problème multi-optima : mon seigneur, envoyez à un endroit idéal (b)

Les méthodes à direction de descente ont l'avantage de converger relativement rapidement vers un point stationnaire en utilisant les informations du gradient. Ces méthodes donc sont très favorables pour un problème convexe unimodal grâce à leur vitesse de convergence (*Figure 1.1a*). Elles nécessitent que la fonction coût soit au moins une fois différentiable par

rapport au paramètres à optimiser ce qui est un problème délicat pour des codes complexes tels qu'en 3D instationnaire. Elles ne permettent d'atteindre que le minimum le plus proche du point de départ, c'est-à-dire un minimum local. Le résultat dépend donc fortement du jeu de paramètres servant à l'initialisation (*Figure 1.1b*).

On peut aussi les utiliser pour accélérer la vitesse de convergence d'algorithmes globaux (comme les algorithmes évolutionnaires) lorsque ces algorithmes ont du mal à converger dans la région proche de l'optimum global.

Les méthodes à direction de descente ne sont donc pas le choix idéal pour résoudre des problèmes d'optimisation multi optima comme celui présenté sur la *Figure 1.1b*. Pour résoudre un tel problème, il vaut mieux utiliser des algorithmes plus globaux.

I.3. METHODES D'ORDRE 0 ET ALGORITHMES DE MINIMISATION GLOBALE

Les méthodes d'ordre 0 ne nécessitent pas le calcul du gradient de la fonction coût. Les algorithmes les plus connus de cette catégorie sont ceux qui s'inspirent de la nature, tels que les algorithmes évolutionnaires, l'algorithme du recuit simulé, l'algorithme de la colonie des fourmis, la méthode de recherche aléatoire, la méthode du simplex, les méthodes de surface de réponse. La plupart de ces algorithmes sont des méthodes d'optimisation globales, sauf celui du simplex.

Avec ce type d'algorithmes, on cherche à générer un ou plusieurs nouveaux points, plus proches de l'optimum, uniquement à partir de la connaissance de la valeur de la fonction coût Φ d'un ou plusieurs points de l'espace des paramètres.

I.3.1. Algorithmes évolutionnaires

Les Algorithmes Evolutionnaires (AE) constituent une discipline impliquant la simulation par un ordinateur du processus de l'évolution naturelle. Ils sont inspirés de la génétique et des mécanismes de la sélection naturelle basés sur la théorie de l'évolution de Darwin, selon laquelle la vie est une compétition où seuls les mieux adaptés survivent et se reproduisent. Ils empruntent les paradigmes de l'évolution biologique tels que la sélection, le croisement et la mutation pour chercher la solution du problème. Les AE utilisent la notion de "population d'individus", dans laquelle chaque individu représente une solution potentielle de l'espace de recherche du problème donné.

Ce sont des méthodes d'optimisation globales. Leur robustesse et leur souplesse permettent d'aborder les problèmes les plus raides. De plus, leur capacité à travailler sur des espaces de recherche non standards (non continus) ainsi que leur faible besoin d'information sur le problème (seulement la fonction coût) offrent les perspectives les plus originales et un large champ d'application. Ils ont donc été appliqués avec succès à de nombreux problèmes où les algorithmes classiques d'optimisation sont incapables de produire des résultats satisfaisants. En outre, en tant qu'algorithme à base de population, leur parallélisation est aisée : il suffit de distribuer l'évaluation de la fonction coût sur autant de processeurs que d'individus de la

population. Leur principal inconvénient est leur coût. Ils nécessitent en effet un grand nombre d'évaluations pour aboutir à l'optimum : c'est le prix qu'ils doivent payer au fait de ne pas utiliser d'autre information sur la fonction et de s'appliquer à de très larges classes de problèmes, aussi chaotiques soient ils.

Nous allons maintenant présenter de manière plus détaillée ces algorithmes. Commençons par leur historique et leurs domaines d'application.

I.3.1.1. Historique et domaines d'application

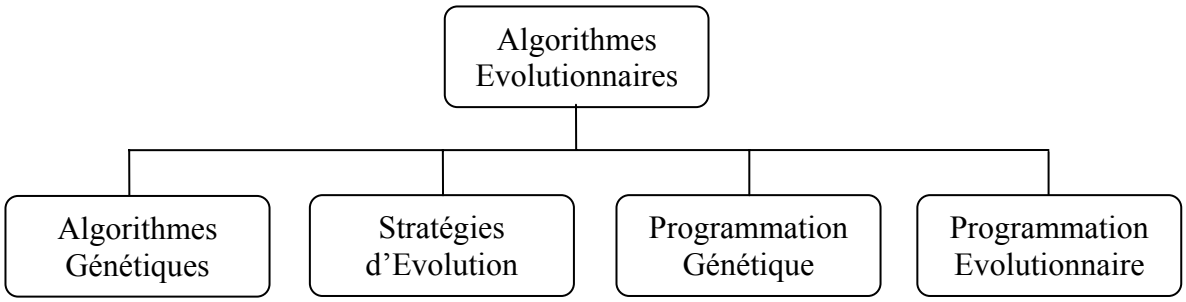

Figure 1.2. Principales catégories des Algorithmes Evolutionnaires

Historiquement, les AE ont été élaborés depuis les années soixante. En général, ils sont divisés en quatre catégories principales, comme présentés dans la *Figure 1.2*.

I.3.1.1.1. Algorithmes Génétiques (AG)

Les AG ont été mis au point par Holland dans les années 60 [*Holland 1962*], [*Holland 1975*] aux Etats-Unis. Ils sont ensuite raffinés et popularisés par De Jong [*De Jong 1975*], Grefenstette [*Grefenstette 1987*], Goldberg [*Goldberg 1989*]. Ces algorithmes s'appuient avant tout sur une représentation binaire des individus.

I.3.1.1.2. Stratégies d'Evolution (SE)

Les premiers efforts pour la mise en place des Stratégies d'Evolution (SE) ont eu lieu dans les années 60 en Allemagne par Rechenberg [*Rechenberg 1965*] [*Rechenberg 1973*] et Schwefel [*Schwefel 1981*]. Ces algorithmes s'appuient sur une représentation en nombres réels et de dimension fixe des individus, ainsi que sur un opérateur de mutation gaussienne. Les SE les plus performantes utilisent les mutations auto adaptatives, dans lesquelles chaque individu porte avec lui les paramètres de la mutation gaussienne qui lui sera appliquée – paramètres eux-même soumis à mutation [*Bäck 1995*].

I.3.1.1.3. Programmation Génétique (PG)

Proposée par Cramer [*Cramer 1985*], la Programmation Génétique (PG) a surtout été popularisée par Koza au début des années 90 [*Koza 1992*], [*Koza 1994*], [*Koza et al. 1999*], [*Banzhaf et al. 1998*]. Elle s'intéresse à l'évolution de programmes. Elle propose un paradigme permettant la programmation automatique d'ordinateurs par des heuristiques

basées sur les mêmes principes d'évolution que les AG. La différence entre la PG et les AG réside essentiellement dans la représentation des individus. En effet, la PG consiste à faire évoluer des individus dont la structure est similaire à celle des programmes informatiques. Koza représente les individus sous forme d'arbres, c'est-à-dire de graphes orientés et sans cycle, dans lesquels chaque noeud est associé à une opération élémentaire relative au domaine du problème. La PG est particulièrement adaptée à l'évolution de structures complexes de dimensions variables.

I.3.1.1.4. Programmation Evolutionnaire (PE)

La Programmation Evolutionnaire (PE) est introduite dans les années 60 par L. Fogel [*Fogel 1964*], [*Fogel et al. 1966*], puis étendue par Burgin [*Burgin 1973*], Atmar [*Atmar 1976*], D.B. Forgel [*Forgel 1992*] et d'autres. Elle a été conçue dans le but de faire évoluer des machines à états finis, puis a été étendue aux problèmes d'optimisation de paramètres. Cette approche met l'emphase sur la relation entre les parents et leurs descendants plutôt que sur les opérateurs génétiques. Elle a été utilisée dans de nombreux autres champs d'application. Les caractéristiques de l'évolution sont très proches de celles des SE. Contrairement aux trois autres AE classiques, la PE n'utilise pas une représentation spécifique des individus mais plutôt un modèle évolutionnaire de haut niveau, qui est associé à une représentation et à un opérateur de mutation directement appropriés au problème à résoudre.

I.3.1.1.5. Champs d'application

Les champs d'application des AE sont très vastes : en économie [*Vallée al. 2001*], en finance, en optimisation de fonctions numériques difficiles (discontinue, multimodales, bruitées) [*De Jong 1980*], en traitement d'image (alignement de photos sattelites, reconnaissance de suspects), en théorie du contrôle optimal, ou encore en théorie des jeux répétés et différentiels, en mécanique des structures [*Burczynski et al. 2001*], [*Chen 2001*], [*Papadrakakis et al. 2001*], en optimisation de forme [*Annicchiarico et al. 1999*], [*Antonio et al. 2002*], [*Chung et al. 1997*], [*Mori et al.1996*], etc. Pour plus d'information sur les applications des AE, le lecteur peut se référer à [*Oduguwa et al. 2005*]

I.3.1.2. Vocabulaire associé

Comme les algorithmes évolutionnaires sont développés à partir de raisonnements issus de la biologie, les termes utilisés en gardant les dénominations. Pour éviter toute confusion de langage, il est nécessaire, suivant en cela Lutton [*Lutton 1999*], de leur préciser le vocabulaire. L'équivalence entre les termes biologiques et les termes d'optimisation est présentée dans le tableau *Tableau 1.1*.

En plus de ce vocabulaire, il nous faut encore distinguer entre le "Génotype" et le "Phénotype". On parle de génotype pour tout ce qui concerne les chromosomes, tandis que les solutions (les vecteurs de l'espace de recherche) constituent le phénotype. Les AE travaillent donc au niveau du génotype.

Algorithmes Evolutionnaires	Méthodes d'optimisation
Individu	Solution potentielle de l'espace de recherche (vecteur des paramètres)
Chromosome	Solution codée (à partir d'une variable binaire, réelle, discrète, etc.)
Gène ou Allèle	Partie composante d'un individu (d'un chromosome)
Population	Ensemble fini (de taille N) d'individus
Performance (ou fonction coût)	Mesure de la qualité des individus basée sur la valeur de la fonction coût et permettant de comparer les individus entre eux afin de déterminer les plus et moins aptes
Evaluation d'un individu	Calcul de la performance d'un individu
Croisement (ou recombinaison)	Opérateur de reproduction appliqué aux individus de la population et qui consiste à échanger ou combiner des composantes entre plusieurs individus.
Mutation	Opérateur de modification d'un ou plusieurs gènes d'un individu dans le but d'introduire une nouvelle variabilité dans la population
Sélection	Processus du choix des individus utilisés pour la reproduction basé sur leur performance
Environnement	Espace de recherche
Remplacement	Processus de formation d'une nouvelle population à partir de l'ensemble des parents et des enfants, effectué le plus souvent sur la base de leur performance
Evolution	Un processus itératif de recherche d'un (ou plusieurs) individu optimal
Génération	Repère le moment de l'évolution

Tableau 1.1 : Equivalence entre « Termes biologiques » et « Termes d'optimisation »

I.3.1.3. Principe de fonctionnement des AE

Le principe de fonctionnement des AE est extrêmement simple, et est présenté par l'organigramme de la *Figure 1.3*.

Pour résoudre un problème d'optimisation, on part d'un ensemble (population) de solutions potentielles (individus) arbitrairement choisies. Les performances de tous les individus de cette population sont évaluées. Sur cette base, l'application des trois opérateurs évolutionnaires de "sélection, croisement et mutation" permet de créer un nouvel ensemble d'individus, appelé "population des enfants". Cette nouvelle population doit être évaluée à son tour afin de décider les quels des enfants méritent de remplacer certains parents et de faire partie de la génération suivante. Le test d'arrêt est ensuit effectué. Si les critères sont vérifiés, on s'arrête. Autrement, on recommence le cycle jusqu'à satisfaction de ces critères.

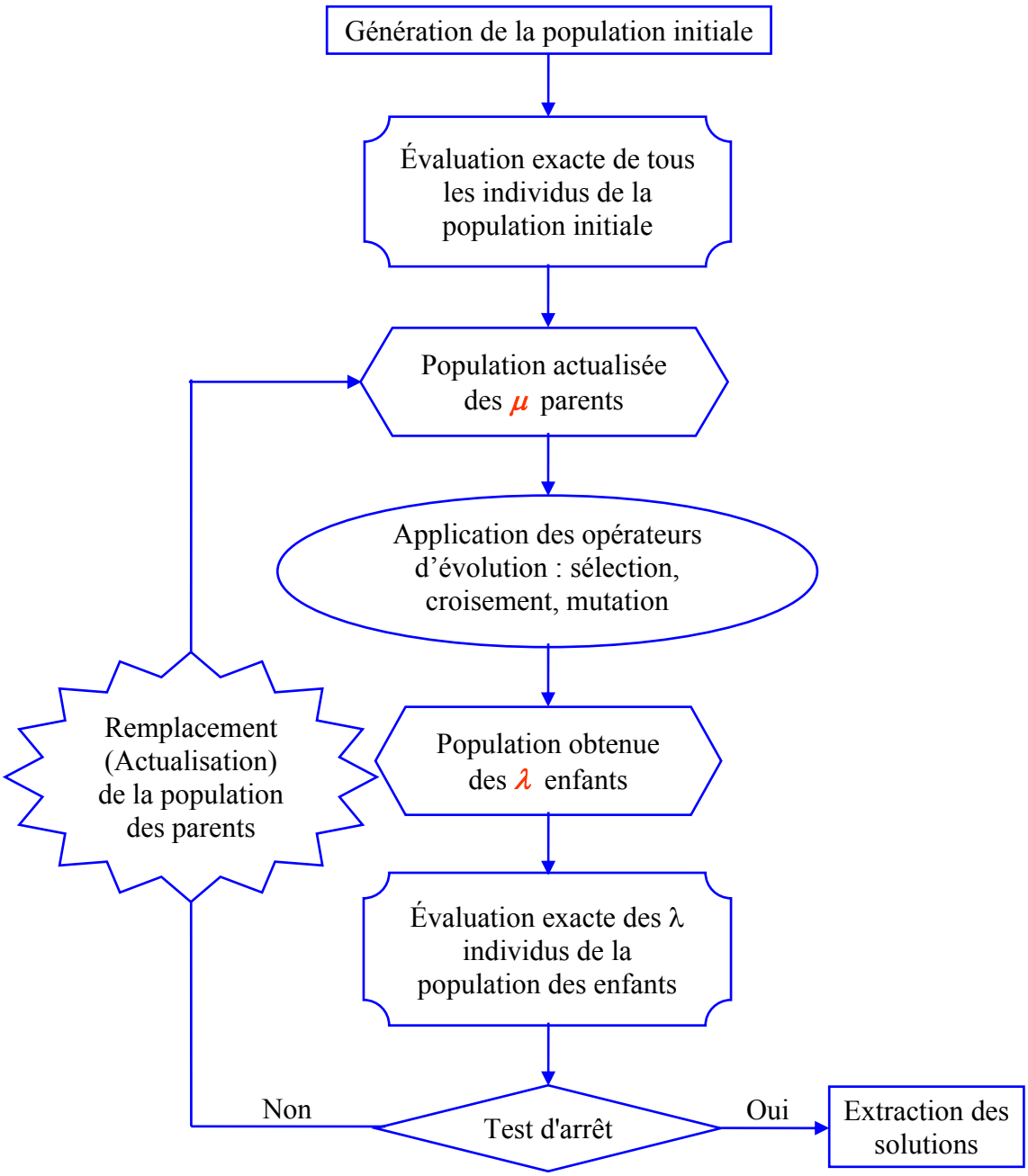

Figure 1.3. Organigramme canonique d'un algorithme évolutionnaire

Le critère d'arrêt des AE peut être la convergence de l'ensemble des solutions vers le même extremum ou quand le meilleur individu de la population atteint un seuil de performance fixé. Le plus souvent le processus est arrêté au bout d'un nombre d'itérations fixé a priori. Notons que la plupart des AE travaillent avec une population de taille fixe, ce qui n'est pas le cas dans la nature.

Nous allons maintenant expliquer en détails les mécanismes de fonctionnement des AE en présentant donc les AG et les SE comme nous nous en servirons pour résoudre nos problèmes d'optimisation. Pour plus de détails sur la programmation génétique et celle évolutionnaire, le lecteur peut se référer aux ouvrages mentionnés dans la partie *I.3.1.1*.

I.3.1.4. Mécanismes de fonctionnement des AE

Pour faire fonctionner les opérateurs génétiques, les AE utilisent un codage (ou représentation) des paramètres à la place des paramètres eux-mêmes. Plusieurs codages ont été développés par différents auteurs comme le codage binaire, le codage réel, le codage de Gray, le codage de permutation, le codage d'état fini, le codage de type «arborescentes ». Les deux codages les plus utilisés sont ceux binaire (utilisé par les AG) et réel (utilisé par les SE).

I.3.1.4.1. Algorithmes génétiques

❖ **Représentation**

Les AG traditionnels utilisent le codage binaire comme représentation des solutions. Chaque individu est représenté par un vecteur binaire (ou chaîne de bits), dont chaque élément prend la valeur 0 ou 1. Ce vecteur est une concaténation des paramètres à optimiser, chaque paramètre étant transformé en une série binaire. La *Figure 1.4* présente un exemple du codage binaire d'une solution avec 3 paramètres. Chaque paramètre est représenté par une série binaire de 4 chiffres $\{0,1\}$.

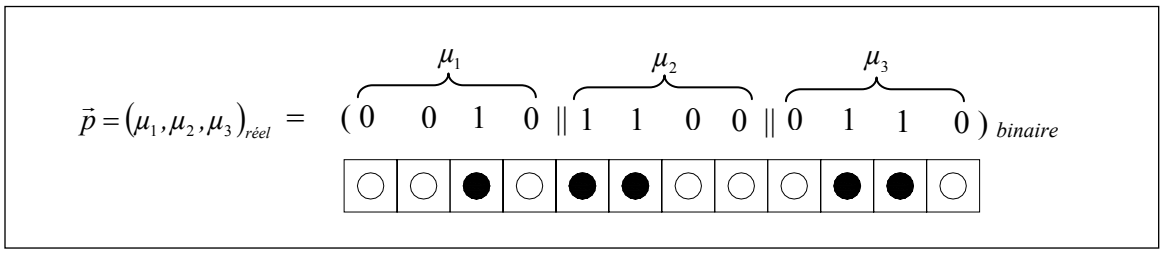

Figure 1.4. Exemple du codage binaire d'une solution potentielle avec 3 paramètres

Cette représentation s'adapte bien aux problèmes où les paramètres ont une représentation binaire canonique, comme les problèmes booléens. Elle s'applique aussi aux problèmes d'optimisation paramétrique continus ($\Phi : \Re^n \to \Re$), mais il est alors nécessaire de définir une technique de codage adéquate de $\Re^n$ vers $\{0,1\}^l$.

La précision du codage dépend du nombre de bits (et donc de la précision de la solution trouvée), plus *l* est grand, plus c'est précis, plus la convergence est longue.

❖ **Sélection des parents**

La sélection est un opérateur clé sur lequel repose en partie la qualité d'un algorithme génétique. Dans cette étape, les chromosomes de la population actuelle sont sélectionnés pour être les parents de la génération suivante. En accord avec la théorie de l'évolution de Darwin, les meilleurs individus doivent survivre et en créer les nouveaux. Il existe plusieurs méthodes pour choisir les meilleurs individus, par exemple la sélection proportionnelle, la sélection par tournoi, la sélection par rang, la sélection selon l'état d'équilibre, etc. Parmi celles-ci, la sélection proportionnelle et la sélection par tournoi sont les méthodes les plus utilisées. On présente ici les méthodes les plus courantes :

↳ *Sélection proportionnelle (Roue de loterie – Roulette selection)*

La méthode de sélection « Tirage à la roulette» introduite par [*Goldberg 1989*] est la méthode la plus connue et la plus utilisée. C'est une méthode stochastique qui reproduit une roulette de casino qui compterait autant de cases que d'individus dans la population. La largeur de la case d'un individu $\vec{x}_i$ est proportionnelle à sa performance $f(\vec{x}_i)$ et prend la valeur $\frac{f(\vec{x}_i)}{\sum_{j=1}^{N} f(\vec{x}_j)}$. La roue est lancée, l'individu sélectionné est désigné par l'arrêt de la roue sur sa case. Pour un problème de maximisation, la performance est la valeur de la fonction coût, pour un problème de minimisation, la performance est l'inverse de la valeur de fonction coût.

L'espérance n_i de la sélection d'un élément $\vec{x}_i$ de la population courante est donnée par l'expression :

$$n_i = \frac{N}{\sum_{j=1}^{N} f(\vec{x}_j)} f(\vec{x}_i) \tag{1.15}$$

où N est le nombre d'individus parents.

L'espérance maximale Max(n_i) de l'ensemble des individus de la population est appelée la pression sélective.

Cette méthode favorise les meilleurs individus mais tous les individus ont des chances d'être choisis. Néanmoins, la méthode peut causer une perte de la diversité de la population si la pression sélective (ou l'espérance n_i du meilleur individu) est élevée. De plus, sa variance est élevée.

Pour diminuer la variance, Baker a proposé en 1987 la méthode de sélection à la roulette avec reste stochastique [*Baker 1987*]. L'approche est similaire à celle de Goldberg, mais cette fois, les individus sélectionnés sont désignés par un ensemble de points équidistants. Le nombre effectif de sélections de l'individu $\vec{x}_i$ sera la partie entière inférieure ou supérieure de son espérance n_i.

↳ *Sélection par tournoi*

Une sélection par tournoi consiste à sélectionner un sous-ensemble de la population, et à ne conserver que le meilleur individu du sous-ensemble. L'opération recommence jusqu'à l'obtention du nombre d'individus requis. Au cours d'une génération, il y a autant de tournois que d'individus à sélectionner. La pression de sélection est ajustée par le nombre q de participants à un tournoi. Un q élevé conduit à une forte pression de sélection.

L'avantage de cette technique est qu'elle est paramétrable par la valeur de q, et peu sensible aux erreurs sur Φ. Par contre, sa variance est élevée [*De Jong et al. 1995*].

Une variante de la sélection par tournoi est le tournoi de Boltzmann [*Maza et al. 1993*], qui assure que la distribution des valeurs d'adaptation d'une population soit proche d'une distribution de Boltzmann. Pour une compétition entre une solution courante $\vec{x}_i$ et une solution alternative $\vec{x}_j$, $\vec{x}_i$ gagne avec la probabilité $\frac{1}{1+e^{\left(\frac{\Phi(\vec{x}_i)-\Phi(\vec{x}_j)}{T}\right)}}$, où T est la température de sélection.

↳ *Sélection par rang*

C'est une sélection qui n'est pas directement proportionnelle à la valeur de la fonction mérite, mais dépend plutôt du rang de l'individu dans la population. Le classement par rang a principalement pour but d'éviter que les individus avec des fonctions de mérite très élevées ne perturbent le processus stochastique en obtenant un nombre de sélections trop important.

↳ *Sélection par troncature*

Cette sélection consiste à choisir de manière déterministe les T% meilleurs individus d'une génération pour générer la suivante. Pour plus de détails sur ces méthodes de sélection et pour les autres méthodes de sélection, on peut se référer à la partie C.2 de [*Bäck et al. 1997*].

❖ **Remplacement**

L'étape de remplacement sert à déterminer quels individus parmi les parents de la génération courante et leurs enfants, seront les parents de la génération suivante. A la différence de l'étape de sélection, durant laquelle des individus peuvent être sélectionnés plusieurs fois, lors de l'étape de remplacement, un individu est ici sélectionné une fois – et il survit alors à la génération suivante – ou pas du tout et il disparaît définitivement de l'évolution en cours. Plusieurs stratégies de remplacement sont présentées dans la littérature pour les AG :

↳ *Remplacement générationnel*

La nouvelle population est composée uniquement des enfants. On fait disparaître tous les individus de la population courante. L'inconvénient majeur de cette approche est la perte de meilleur individu, si on ne le conserve pas systématiquement dans la nouvelle population.

↳ *Remplacement élitiste*

La nouvelle génération garde certaines "bonnes" solutions (sélection élitiste) de la génération courante et est complétée par des enfants.

↳ *Remplacement continu*

Des enfants, choisis aléatoirement, remplacent de façon régulière les individus les moins performants de la génération courante.

❖ **Croisement**

Le croisement est l'opérateur principal des AG. C'est un opérateur génétique relatif à plusieurs individus parents (souvent deux). Son rôle consiste à combiner les génotypes des individus pour en produire un nouveau. Il fait partie du mécanisme de convergence de l'AG, qui permet de concentrer la population autour des meilleurs individus. On distingue plusieurs types de croisements possibles. Les plus utilisés sont :

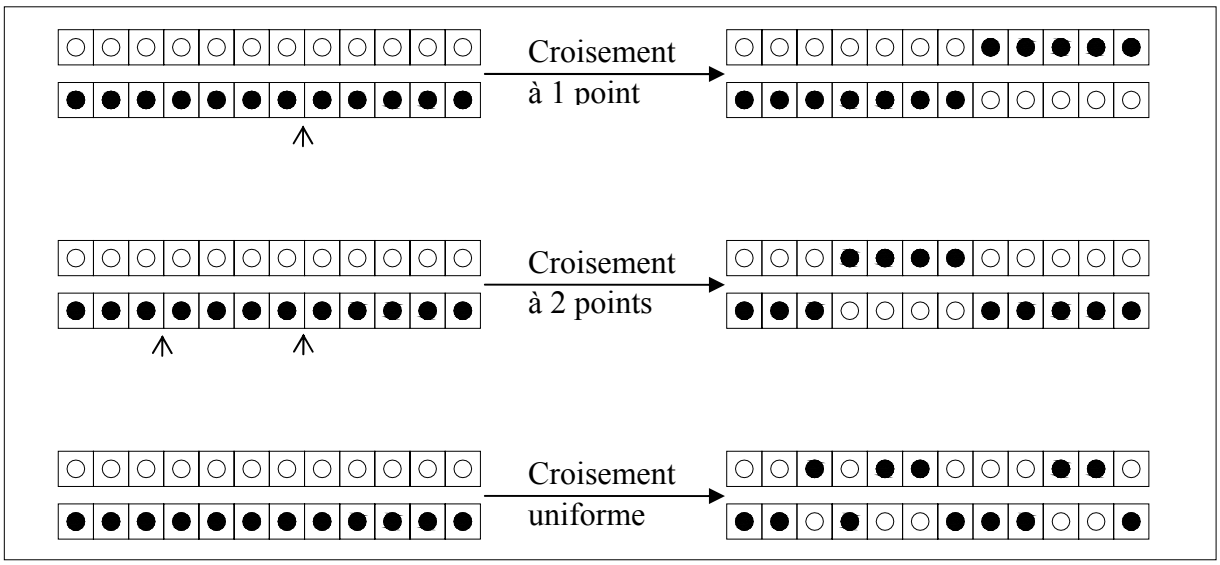

Figure 1.5. Méthodes de recombinaison (croisement) utilisées par l'AG

- *Croisement à 1 point* [*Holland 1962*]

Le croisement à un point est l'opérateur de croisement le plus simple et le plus classique. Il consiste à choisir aléatoirement un point de coupure, puis à subdiviser le génotype de chacun des parents en deux parties de part et d'autre de ce point. Les fragments obtenus sont alors échangés pour créer les génotypes des enfants (*Figure 1.5*).

- *Croisement à multipoints* [*De Jong et al. 1991*]

Le croisement multipoints est une généralisation du croisement à un point. Au lieu de choisir un seul point de coupure, on en sélectionne *k*, aléatoirement. Dans le croisement multipoints, les points de coupure sont fixés par avance. La *Figure 1.5* représente un croisement multipoints (deux points dans l'exemple).

- *Croisement uniforme* [*Syswerda 1989*]

Ce type de croisement est la généralisation du croisement multipoints. Dans le croisement uniforme, chaque gène d'un enfant est choisi aléatoirement entre les gènes des parents ayant la même position dans le chromosome, avec une probabilité de 0,5 s'il y a deux parents. Le second enfant est construit en prenant les choix complémentaires du premier enfant. Un exemple du croisement uniforme est aussi présenté sur la *Figure 1.5*.

❖ Mutation

Les AG utilisent l'opérateur de mutation comme moyen de préserver la diversité de la population. La mutation utilisée est binaire. Elle inverse aléatoirement les bits du génotype, avec une faible probabilité, typiquement de 0,01 à 0,001. La *Figure 1.6* nous montre un exemple de la mutation binaire.

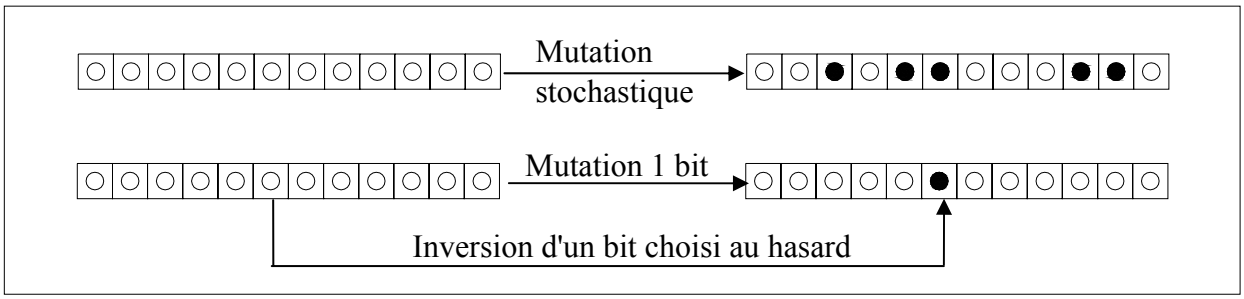

Figure 1.6. Méthodes de mutation utilisées par le AG

- *Mutation stochastique (bit flip)* :

Etant la plus employée avec le codage binaire, cette méthode de mutation consiste à inverser indépendamment chaque bit du chromosome. Un test sur le taux de mutation est effectué pour chacun des bits du chromosome : en cas de succès, le bit testé est alors inversé.

- *Mutation 1 bit*

Un bit du chromosome est choisi au hasard. Sa valeur est alors inversée.

I.3.1.4.2. Stratégies d'Evolution

Contrairement aux AG, la représentation utilisée par les SE est celle de vecteurs à valeurs réelles des paramètres ($\vec{p} \in \Re^n$) lorsqu'elles sont utilisées pour des problèmes d'optimisation continus. Rechenberg [Rechenberg 1994] a proposé d'associer à chaque valeur de la solution un vecteur auxiliaire qui détermine comment faire varier cette solution [*Schwefel 1981*] et permet l'auto-adaptation. Un individu consiste donc maintenant en le vecteur des paramètres d'objet $\vec{p}$ et celui de paramètres stratégiques $\vec{\sigma}$ - l'écart type de mutation :

$$\vec{a} = (\vec{p}, \vec{\sigma}) \tag{1.16}$$

❖ **Sélection et Remplacement**

La sélection d'un ou plusieurs parents pour générer les nouveaux individus dans les SE est semblable à celle des AG. En revanche, le remplacement est complètement déterministe, ce qui lui donne un rôle clef dans l'évolution en guidant la recherche vers les meilleurs individus. Pour produire les nouveaux individus, il opère en sélectionnant les μ ($1 < \mu < \lambda$) premiers individus par ordre de performance et remplace les anciens parents par :

- l'union de μ parents et λ enfants : schéma appelé stratégie $(\mu + \lambda)$
- l'ensemble des λ enfants : schéma appelé stratégie (μ, λ)

La stratégie ($\mu+\lambda$) est élitiste. Elle garantie une amélioration monotone de la performance de la population, mais elle peut converger prématurément car elle s'adapte mal à un éventuel changement d'environnement. En revanche, avec la stratégie (μ , λ), la valeur de la meilleure performance peut décroître si tous les enfants sont inférieurs aux parents, mais l'algorithme est plus flexible vis-à-vis des changements d'environnement. De plus, la régression des meilleures performances peut aider le processus de recherche à sortir des régions d'attraction des optima locaux pour aller explorer ailleurs.

❖ Croisement

Le croisement est facultatif pour les SE. Il n'est pas obligatoirement présent dans la génération des nouveaux individus. Avec le codage réel, le croisement qui échange les informations entre les parents peuvent sembler le même que dans le cas du codage binaire des AG, mais il existe une différence fondamentale [*Séfrioui 1998*] : avec le codage réel, le point de coupure tombe nécessairement entre deux composantes du vecteur des paramètres alors que pour le codage en binaire le point de coupure peut tomber à l'intérieur d'une composante. On peut aussi réaliser des croisements intermédiaires comme présentés dans la formule (1.17). La recombinaison peut être différente des paramètres d'objet et pour les paramètres stratégiques. Un exemple de règles de recombinaison pour le vecteur des paramètres d'objet $\vec{p}=(p_1,p_2,...,p_i,...,p_N)$ est présenté dans [*Bäck et al. 1993*]:

$$p'_i=\begin{cases} p_{S,i} & \textit{sans recombinaison} \\ p_{S,i} \;\; ou \;\; p_{T,i} & \textit{recombinaison discrète} \\ p_{S,i}+\chi.(p_{T,i}-p_{S,i}) & \textit{recombinaison intermédiaire} \\ p_{S_i,i} \;\; ou \;\; p_{T_i,i} & \textit{recombinaison discrète globale} \\ p_{S_i,i}+\chi_i.(p_{T_i,i}-p_{S_i,i}) & \textit{recombinaison intermédiaire globale} \end{cases} \qquad (1.17)$$

Les indices *S* et *T* représentent les deux individus choisis aléatoirement dans la population des μ parents. $\chi\in[0,1]$ est une variable aléatoire uniforme; la valeur $\chi=1/2$ est souvent utilisée. Avec la recombinaison globale, pour chaque composante p'_i de l'individu enfant $\vec{p}'$ les parents S_i, T_i et la variable χ_i sont déterminés indépendamment. Empiriquement, la recombinaison discrète sur les paramètres d'objet et celle intermédiaire sur les paramètres stratégiques semblent donner les meilleurs résultats. La recombinaison sur les paramètres stratégiques s'avère obligatoire pour le mécanisme de « recombinaison » des SE fonctionne [*Bäck et al. 1993*].

❖ Mutation gaussienne auto-adaptative

Contrairement aux AG, l'opérateur de mutation est toujours présent dans l'évolution des SE (tandis que le croisement est facultatif). La mutation garantit la globalité de la recherche : c'est le principal opérateur d'exploration. Les SE peuvent donc fonctionner avec une population d'un seul individu.

Toutefois, lorsque l'opérateur de mutation a une intensité variable (comme c'est le cas pour l'opérateur de mutation gaussienne auto-adaptatif décrit ci-dessous), la mutation peut aussi

être un opérateur d'exploitation. Dans le cadre de génotypes réels, l'opérateur de mutation le plus efficace et le plus utilisé est la mutation gaussienne auto-adaptative, que nous allons maintenant détailler.

Le principe de base de la mutation gaussienne est d'ajouter un bruit gaussien centré $N(0,\sigma)$ aux variables que l'on désire faire muter :

$$p'_i = p_i + N(0,\sigma) \tag{1.18}$$

Tout l'art réside dans le choix du paramètre σ, la variance de la perturbation gaussienne, et définissant donc l'intensité de la mutation suivant une loi normale : environ 67% des tirages seront compris entre $-\sigma$ et $+\sigma$ et plus de 99% entre -3σ et $+3\sigma$. Il y a également une probabilité strictement positive de tirer tout nombre réel positif ou négatif car la "queue" de la distribution ne s'annule jamais. Tous les ingrédients sont donc présents pour pouvoir faire de cette mutation un opérateur soit d'exploration (grandes valeurs de σ) soit d'exploitation (petites valeurs de σ). Il faut par contre trouver la bonne loi de mise à jour de σ pour un bon compromis entre ces deux comportements.

Une première idée fut de faire muter plus fortement les mauvais individus (puisqu'il ne sert à rien de les exploiter, autant explorer davantage) et faiblement les bons individus (pour exploiter l'espace de recherche autour d'eux). Cette idée, utilisée dans les premiers temps de la Programmation évolutionnaire [*Fogel et al. 1966*], se révéla difficile à mettre en oeuvre dans le cadre de variables réelles.

La première approche vraiment adaptative, c'est-à-dire dans laquelle la décision est prise par rapport à la situation courante, fut la célèbre règle des 1/5 de Rechenberg [*Rechenberg 1973*]. La mutation considérée est isotrope. Son principe consiste à augmenter la valeur de l'écart type σ de la mutation lorsque trop de mutations sont réussies, c'est-à-dire lorsque trop d'enfants ont une performance meilleure que celle des parents (trop d'exploitation), et réciproquement de diminuer la valeur de l'écart type σ de lorsque pas assez de mutations sont réussies (trop d'exploration). La règle 1/5 s'utilise de la manière suivante : on commence par se fixer un temps d'observation T (correspondant à un nombre de générations), et toutes les T générations, on calcule le taux τ de mutations réussies. Si τ est supérieur à 0,2 alors σ est augmenté d'un facteur 1,22 , sinon σ est diminué d'un facteur 0,83.

Cette approche possède quelques faiblesses. Elle ne prend pas en compte les caractéristiques locales des performances du fait que la même valeur de σ est utilisée pour toute la population et pour toutes les composantes du génotype. Pour pallier ce défaut, ainsi que pour se débarrasser élégamment de la tâche fastidieuse du réglage des paramètres de la mutation, Rechenberg [*Rechenberg 1973*] et Schwefel [*Schwefel 1981*] ont proposé de rendre "la mutation auto-adaptative" : chaque individu possède ses propres paramètres de mutation, qui sont eux-mêmes sujets à mutation, avant d'être utilisés pour la mutation des variables elles-mêmes.

Avec la mutation auto-adaptive, l'étape de remplacement sélectionne les individus avec leurs paramètres de mutation. Les individus qui survivront seront ceux qui auront à la fois les

bonnes valeurs pour les variables du problème (sur lesquelles se fait la sélection), mais aussi ceux qui auront les bonnes valeurs des paramètres de mutation, sinon ils seront immanquablement dépassés par d'autres, mieux adaptés aux caractéristiques locales du paysage de performance. De manière imagée, lorsque le gradient est fort dans une direction, il faut faire de "petites" mutations, et inversement. Avec cette méthode, on constate que :

– des mutations successives avec des paramètres de mutation aberrants ne peuvent pas être constamment réussies

– les individus qui survivent longtemps sont le fruit de nombreuses mutations successives réussies et doivent donc avoir de "bons" paramètres de mutation.

Il existe trois types de mutations auto-adaptatives :

– La mutation isotropie, dans laquelle chaque individu possède un scalaire σ qui est utilisé pour l'ensemble des composantes du vecteur lors de la mutation. Plus précisément, la mutation de l'individu $(\vec{p},\sigma)$ s'effectue en deux temps : on commence par faire muter l'écarte type σ suivant une loi log-normale (pour des raisons de symétrie multiplicative autour de 1), puis par faire muter des variables p_i, en utilisant la nouvelle valeur de σ.

$$\begin{aligned} &\sigma' = \sigma.exp(\tau\, N(0,1)) \\ &\forall i \qquad p'_i = p_i + N(0,\sigma) \end{aligned} \tag{1.19}$$

où τ est un paramètre de la SE.

– La mutation anisotrope, dans laquelle les paramètres de la mutation sont un vecteur de valeurs qui représente les écarts types dans chacune des directions canoniques. La mutation s'effectue aussi en deux temps,

$$\begin{aligned} &\forall i \qquad \sigma'_i = \sigma_i.exp(\tau\, N(0,1) + \tau' N_i(0,1)) \\ &\forall i \qquad p'_i = p_i + N(0,\sigma_i) \end{aligned} \tag{1.20}$$

où τ et τ' sont aussi des paramètres (de second ordre) de la SE.

– Les mutations corrélées dans lesquelles les paramètres de la mutation sont une matrice de covariance complète, et que nous ne détaillerons pas ici, le lecteur peut se référer à [*Auger 2004*] pour plus de détails.

Les valeurs préconisées par Schwefel [*Schwefel 1981*], basées sur des études théoriques de la fonction sphère en grande dimension, sont les suivantes :

$$\tau \propto \frac{1}{\sqrt{2n}} \qquad \tau' \propto \frac{1}{\sqrt{2\sqrt{n}}} \tag{1.21}$$

I.3.2. Méthodes de surface de réponse

Le plus souvent les méthodes de surface de réponse sont des méthodes d'optimisation basées sur les plans d'expériences. A l'origine, les plans d'expériences sont créés pour s'appliquer à l'expérimentation. Grâce à eux, l'expérimentateur peut répondre aux questions « comment sélectionner les expériences à faire, quelle est la meilleure stratégie » pour :

- Aboutir le plus rapidement possible aux résultats espérés avec une bonne précision, en évitant des expériences inutiles
- Conduire à la modélisation et à l'optimisation des phénomènes étudiés

Une littérature abondante existe sur les plans d'expériences, mais dans le cas d'expérimentation numérique, tous les aspects liés aux erreurs de mesure sont sans objet.

Dans le domaine d'optimisation numérique, un plan d'expériences peut être utilisé comme un support ou une étape préliminaire à l'optimisation par les méthodes de surface de réponse. Grâce au plan d'expériences, nous pouvons éviter les évaluations de la fonction coût inutiles et économiser le temps de résolution.

Le principe des méthodes d'optimisation par surface de réponse consiste à remplacer la résolution du problème d'optimisation réel par celle de problèmes approchés. Le schéma général de la résolution du problème d'optimisation par les méthodes de surface de réponse est présenté sur la *Figure 1.7*.

Grâce à une base de données composée de plusieurs points (solutions) déjà évalués, on approxime la fonction Φ (ainsi que les contraintes et le gradient) sur l'espace des paramètres d'optimisation par des fonctions mathématiques. Ces approximations conduisent donc au problème d'optimisation approché (1.22):

$$\begin{cases} Minimiser\, \Phi^{appr}(p) \\ c_i^{appr}(p) \le 0 \quad \forall i = 1,...,m_i \\ h_i^{appr}(p) = 0 \quad \forall i = 1,...,m_e \\ p \in \Re^n \end{cases} \tag{1.22}$$

Le problème d'optimisation approché (1.22) pourrait être résolu avec tous les types d'algorithmes d'optimisation comme ceux à direction de descente, les algorithmes évolutionnaires, les algorithmes hybrides, etc. Quelque soit l'algorithme retenu, le temps de la résolution du problème (1.22) est négligeable devant le celui de la résolution du problème réel (en une milliseconde, les ordinateurs peuvent réaliser facilement mille évaluations de la fonction coût ainsi que des contraintes et le gradient du problème approché). La création de la base de données initiale ou l'échantillonnage (Design of Experiments – DOE en anglais) et la méthode pour approximer la fonction objectif sont les éléments principaux qui caractérisent les méthodes de surface de réponse. La qualité des solutions obtenues est en grande partie fonction de ces éléments. Une description détaillée des plans d'expériences et les méthodes d'approximation est décrite dans l'ouvrage [*Myers et al. 2002*].

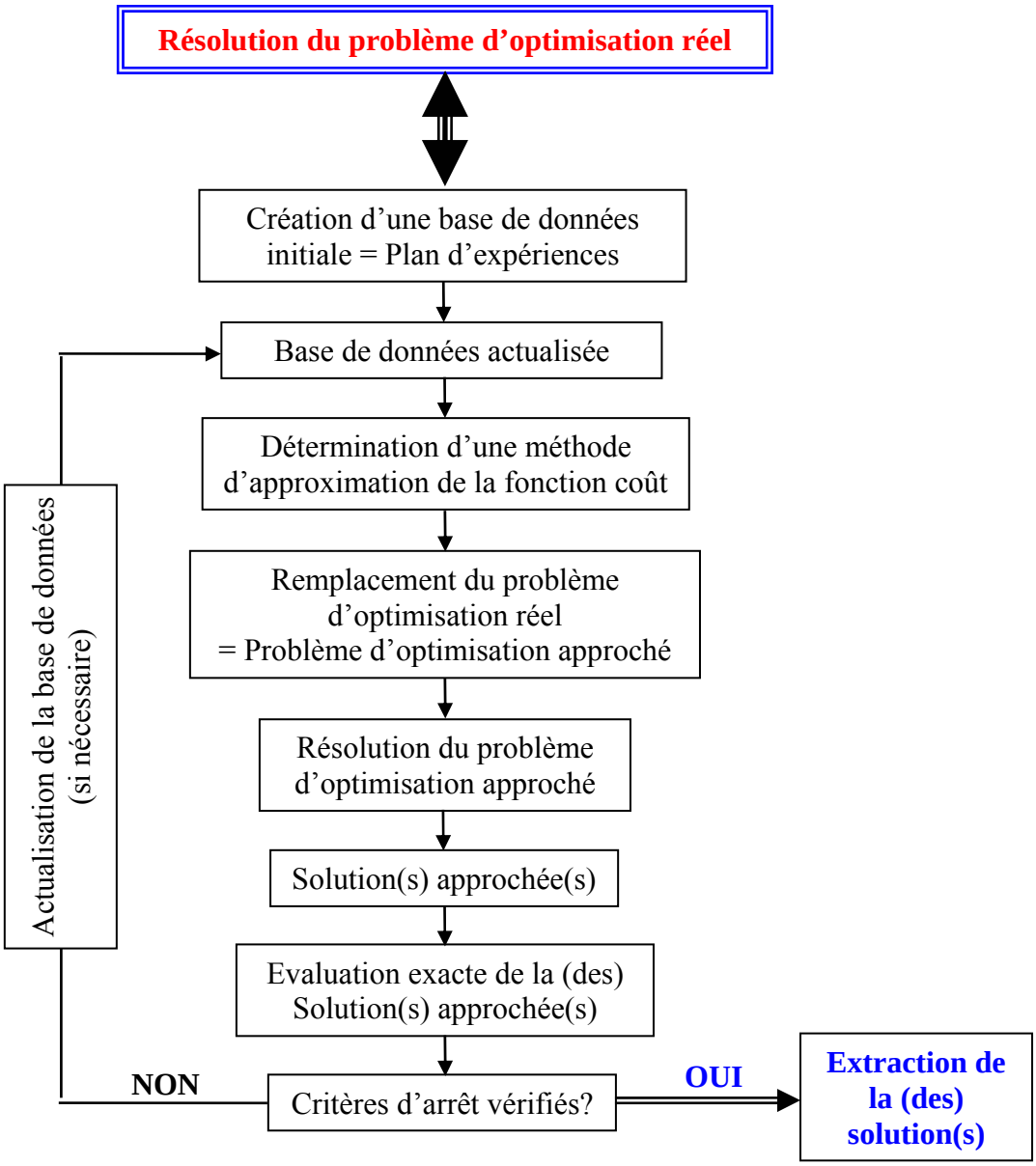

Figure 1.7. Résolution d'un problème d'optimisation par la méthode des surfaces de réponse

Dans le domaine de la mise en forme, les méthodes de surface de réponse ont été employées pour résoudre différents problèmes d'optimisation.

Pour l'optimisation des procédés de mise en forme des métaux, *Bonte et al.* [*Bonte et al. 2005a*], [*Bonte et al. 2005b*] et [*Bonte 2005*] ont utilisé une méthode de surface de réponse et ont obtenu des résultats très satisfaisants. A chaque itération d'optimisation, *Bonte et al.* ont créé une nouvelle base de données en utilisant la méthode d'échantillonnage "Latin Hypercube Design" (LHD) [*Santner et al. 2003*] [*Martin et al. 2004*] et le plan factoriel complet. La fonction coût (ainsi que les contraintes et les gradients) est approximée grâce à cette base de données. Pour faire cela, ils construisent sept méta-modèles à base de différentes régressions polynomiales et de la méthode de krigeage. Puis, un test de précision est effectué pour ces méta-modèles afin d'identifier le meilleur. Une fois que l'approximation de fonction coût réalisée, ils ont utilisé l'algorithme SQP (Sequential Quadratique Programming) [*Haftka et al. 1992*] pour obtenir la solution approchée optimale. Afin d'éviter le fait de tomber dans un

optimum local, ils ont choisi de lancer l'algorithme SQP à partir de chaque point de la base de données. Ils ont pris la meilleure solution approchée parmi celles obtenues par les différentes optimisations. Puis une autre évaluation exacte devait être réalisée pour obtenir la réponse réelle du problème à l'itération d'optimisation. Si la solution optimisée est satisfaisante, ils s'arrêtent. Sinon, ils recommencent l'algorithme. Cependant, à l'étape échantillonnage, ils créent un nouveau plan d'expérience en tenant compte de l'existence des points déjà évalués aux itérations précédentes pour avoir une approximation plus précise.

Une approche similaire à celle présentée précédemment est utilisée par [*Beauchesne et al. 2005*] pour faire l'optimisation du procédé d'hydroformage. Cependant, au lieu de 7 métamodèles différents, seule l'approximation à base de krigeage non linéaire est utilisée pour approcher la fonction coût.

Ayad at al. [*Ayad at al. 2005*] ont aussi utilisé la méthode des surfaces de réponse pour l'optimisation du procédé de ségrégation de poudre lors du moulage par injection métallique. La stratégie utilisée consiste en 3 étapes principales : d'abord utiliser un plan d'expérience de type Taguchi [*Sado et al. 1991*] pour déterminer les paramètres de comportement du matériau et du procédé les plus importants afin de diminuer la dimension du problème d'optimisation. Ensuite, ils construisent une approximation basée sur la méthode des moindres carrés mobiles [*Belytschko et al. 1996*], puis s'en servent pour chercher la solution optimale approchée du problème grâce à un algorithme génétique. Enfin, afin d'améliorer la recherche de l'optimum, et de localiser correctement celui-ci, ils ont développé une méthode adaptative de type moindres carrés mobiles en raffinant l'espace de recherche.

Une approche similaire à celle de Ayad et al. est utilisé par Ben Ayed et al. [*Ben Ayed et al. 2005*] pour l'optimisation des efforts de serre-flan en emboutissage. La méthode des moindres carrés mobiles est utilisée pour approximer la valeur de fonction coût. Un algorithme d'optimisation de type SQP est ensuite utilisé pour résoudre le problème approximé à partir de plusieurs points de départ pour trouver l'optimum global du problème.

Naceur et al. [*Naceur et al. 2004*] ont utilisé une méthode de surface de réponse à base d'approximation diffuse [*Nayroles et al. 1991*] pour faire l'optimisation du procédé d'emboutissage, suivant une approche semblable.

Pour l'optimisation du procédé de pliage (créer les pièces de sécurité), Bahloul et al. [*Bahloul et al. 2005*] ont tenté d'utiliser la méthode d'approximation de type « réseau de neurones » combinée avec un algorithme évolutionnaire. Un plan d'expériences de Taguchi est utilisé pour l'étape d'échantillonnage et pour sélectionner les paramètres les plus importants du problème d'optimisation.

I.3.3. Autres méthodes d'optimisation d'ordre 0

I.3.3.1. Méthodes de recherche aléatoire/probabiliste

La méthode de recherche aléatoire consiste à tirer aléatoirement, à chaque itération un point dans l'espace de recherche. La valeur de fonction objectif Φ est ensuite évaluée en ce point et

comparée à celle du point de départ. Si elle est meilleure, cette valeur est enregistrée, ainsi que la solution correspondante, et le processus continue. Sinon on repart du point de départ et on recommence le procédé, jusqu'à ce que les conditions d'arrêt soient atteintes. Le grand avantage de cette méthode est sa simplicité. Le temps de calcul en constitue une grande faiblesse.

I.3.3.2. Méthodes du simplex

Cette méthode d'ordre 0 déterministe a été introduite par Nelder et Mead [*Nelder et al. 1965*].

Supposons que la fonction coût Φ ait n paramètres. On définit un simplex comme étant une figure géométrique (polygone, triangles, etc.) de volume non nul contenant (n+1) sommets. Donc, à chaque itération de l'algorithme du simplex, (n+1) points sont utilisés pour déterminer un pas d'essai. Les points p_i sont ordonnés de manière à avoir $\Phi(p_1) \leq \Phi(p_2) \leq .... \leq \Phi(p_{n+1})$. Des nouveaux points sont obtenus en utilisant de très simples opérations algébriques, qui se traduisent par des transformations géométriques élémentaires (réflexion, contraction, expansion, et multicontraction appelée aussi rétrécissement), et ces points sont acceptés ou rejetés en fonction de leur valeur de la fonction objectif. Le simplex se transforme, il s'étend, se contracte, à chaque mouvement. Ainsi il s'adapte à l'allure de la fonction, jusqu'à ce qu'il s'approche de l'optimum. A chaque transformation, le plus mauvais point courant x_i est remplacé par le nouveau point déterminé.

La méthode du simplex n'utilise que des valeurs ponctuelles de la fonction coût et ne nécessite pas l'estimation du gradient. Cette méthode peut donc être utilisée pour la recherche du minimum d'une fonction coût non-différentiable. Elle semble efficace tant que le nombre de paramètres est petit [*Kusiak 1989*]. Lorsque le nombre de paramètres est supérieur à trois, elle semble mal adaptée du point de vue du coût, et devient moins intéressante que les méthodes à direction de descente [*Nouatin 2000*] [*Vielledent 1999*] [*Kusiak 1989*].

Ohata et al. [*Ohata et al. 1998*] ont utilisé cet algorithme pour résoudre un problème d'optimisation d'un procédé de mise en forme 3D de plaques, en deux opérations et avec trois paramètres à optimiser. Coupez et al. [*Coupez et al. 1999*] l'ont utilisé pour l'optimisation du profil du champ de vitesse dans un procédé d'injection 3D.

Nakamashi et al. [*Nakamashi et al. 1998*] ont proposé une étude comparative entre la méthode du simplex et la méthode heuristique du recuit simulé [*Aarts 1989*], sur le même cas que Ohata et al., mais pour sept paramètres. De plus, il est à noter que, même si généralement l'algorithme fonctionne bien, il existe des cas où la méthode ne converge pas. Des exemples de stagnation en des points non-stationnaires ont en effet été décrits dans [*Mc Kinnon 1998*] dans des cas de minimisation de fonctions strictement convexes.

I.3.3.3. Méthode du recuit simulé

Cette méthode d'optimisation a été mise au point en 1983 par trois chercheurs de la société IBM, S. Kirkpatrick, C.D. Gelatt et M.P. Vecchi [*Kirkpatrick et al. 1983*].

Le recuit simulé est une méthode d'optimisation stochastique tirant son origine d'un processus thermodynamique. Cette méthode est issue d'une analogie avec le phénomène physique de refroidissement lent d'un corps en fusion, qui le conduit à un état solide de basse énergie. Il faut abaisser lentement la température, en marquant des paliers suffisamment longs, pour que le corps atteigne l'équilibre thermodynamique à chaque palier de température. Pour les matériaux, cette basse énergie se manifeste par l'obtention d'une structure régulière, comme les cristaux dans l'acier. L'analogie exploitée par le recuit simulé consiste à considérer la fonction Φ à minimiser comme fonction d'énergie, et une solution p peut être considérée comme un état donné de la matière dont $\Phi(p)$ est l'énergie. Le recuit simulé exploite généralement le critère défini par l'algorithme de Metropolis et al. [*Metropolis et al. 1953*] pour l'acceptation d'une solution obtenue par perturbation de la solution courante.

Des études théoriques du recuit simulé ont pu montrer que sous certaines conditions, l'algorithme du recuit convergeait vers un optimum global. Ce résultat est important car il nous assure que le recuit simulé peut trouver la meilleure solution, si on le laisse chercher indéfiniment. Les principaux inconvénients du recuit simulé résident dans le choix des nombreux paramètres, tels que la température initiale, la loi de décroissance de la température, les critères d'arrêt ou la longueur des paliers de température. Ces paramètres sont souvent choisis de manière empirique.

I.3.3.4. Algorithmes de colonie des fourmis

Le comportement des insectes sociaux est caractérisé par l'auto-organisation. Les individus communiquent en changeant les propriétés locales de leur environnement, et, par le biais de ce moyen de communication limité, une sorte d'intelligence collective émerge. Marco Dorigo [*Dorigo et al. 1996*] de l'Université Libre de Bruxelles a inventé l'algorithme à colonies de fourmis lorsqu'il a observé des fourmis dans leur chemin de recherche de la nourriture. Celles-ci ont la capacité de trouver le chemin le plus court entre leur nid et une source de nourriture, en contournant les obstacles qui jonchent leur chemin.

L'idée générale de l'algorithme de colonie de fourmis est d'imiter le comportement coopératif d'une colonie de fourmis naturelles à l'aide des fourmis artificielles se déplaçant à travers le graphe qui représente le problème à résoudre. Le principe est le suivant : les fourmis cherchent de la nourriture et se déplacent de façon quasi aléatoire. Tout au long de leur déplacement, elles laissent derrière elles une substance chimique appelée phéromone. Cette substance a la propriété de s'évaporer au cours du temps et a pour but de guider les fourmis vers leur objectif. Une fois cet objectif atteint (dans notre cas, la nourriture trouvée), les fourmis rentrent au nid en empruntant le même chemin qu'à l'aller, grâce à leur trace de phéromone. Celle-ci s'en trouve renforcée. Plus une trace de phéromone est concentrée, plus elle va attirer les fourmis. Au fil du temps, on va donc constater l'émergence du plus court chemin vers la nourriture grâce au renforcement de la trace de phéromone.

Dans la nature, nous pouvons remarquer que les fourmis se déplacent "en ligne", suivant le chemin des fourmis précédentes. Pour mieux comprendre le phénomène et l'algorithme, faisons une simple expérience en plaçant maintenant un obstacle sur cette ligne, de manière à ce que le contournement de l'obstacle par un côté soit nettement plus court que par l'autre

(*Figure 1.8*). Au bout d'un certain temps nous observons que toutes les fourmis contourneront l'obstacle par le côté le plus court.

Le phénomène s'explique de la manière suivante : les fourmis déposent en marchant des marqueurs chimiques appelés phéromones. Les autres fourmis suivent le chemin tracé par ces phéromones. Plus un chemin est marqué, plus elles ont tendance à le suivre. Lorsque l'on pose l'obstacle, le chemin de phéromones est coupé, et les fourmis choisissent au hasard l'un ou l'autre côté pour contourner l'obstacle. Comme un chemin est plus court, plus de fourmis auront franchi l'obstacle en passant par ce côté. Par exemple, sur 6 fourmis, 3 fourmis pourront être passées par le côté court alors que les 3 fourmis étant passées par le côté long n'auront pas encore fini de contourner l'obstacle. Une fourmi arrivant en sens inverse verra donc plus de phéromone sur le chemin qui passe par le côté court, et prendra donc ce chemin.

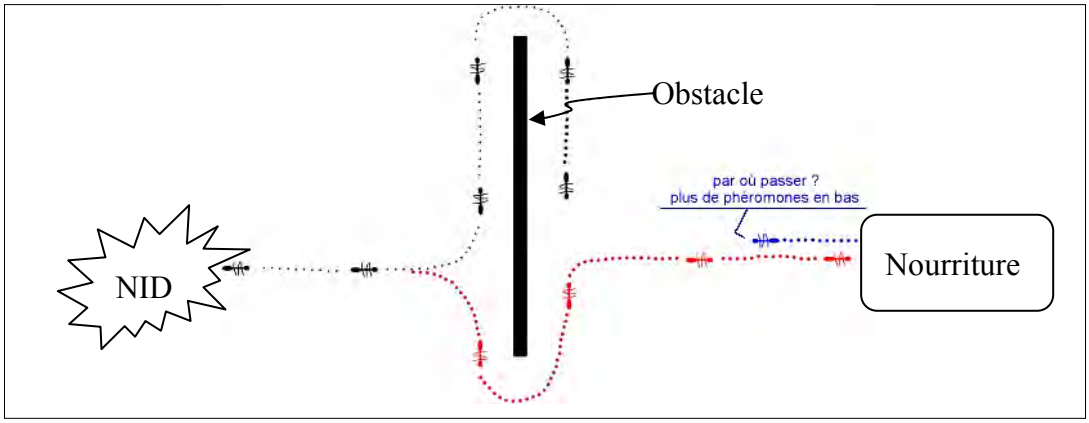

Figure 1.8. Comportement des fourmis lors de franchir l'obstacle

L'algorithme de colonies de fourmis a été à l'origine principalement utilisé pour produire des solutions quasi-optimales au problème du voyageur de commerce, puis, plus généralement, aux problèmes d'optimisation combinatoire. On observe depuis ses débuts que son emploi se généralise à plusieurs domaines, depuis l'optimisation continue jusqu'à la classification ou encore le traitement d'image. On peut trouver une liste des applications de plusieurs variantes de cet algorithme à la page 51 de la thèse de Roux [*Roux 2001*].

I.4. METHODES HYBRIDES

L'hybridation des algorithmes est pour objectif de mélanger de manière harmonieuse deux ou plusieurs méthodes distinctes afin de ne retenir que les caractéristiques les plus intéressantes de chacune de ces méthodes.

L'approche d'hybridation la plus connue est celle entre un algorithme évolutionnaire et un algorithme à direction de descente [*Oulladji et al. 2003*]. Le principe de cette approche d'hybridation est assez simple. Il consiste à lancer une recherche au niveau global avec un AE, puis passer à la recherche locale avec un algorithme à direction de descente pour affiner le résultat. Cela nécessite donc d'effectuer une répartition des tâches. L'AE se charge de détecter les régions de l'espace de recherche qui sont susceptibles de se révéler les plus intéressantes. Puis, l'algorithme à direction de descente prend comme point de départ les

meilleures solutions trouvées par l'AE, et s'attache à les affiner aussi rapidement possible. Pourtant, il est délicat de décider le moment de transition ou à partir duquel l'algorithme à direction de descente doit prendre le relais et faire son travail. En effet, si cela se fait trop tôt, il y a de fortes chances pour que l'algorithme touche sa fin par la convergence vers un optimum local. Au contraire, si la transition se produit trop tard, on perd en temps de calcul car les avantages de l'algorithme à direction de descente ne sont pas pleinement exploités.

L'hybridation peut aussi être réalisée entre un AE et une méthode d'approximation. Avec cette version, la méthode d'approximation est utilisée pour accélérer la convergence de l'AE. L'idée est donc de remplacer la fonction objectif par une fonction approchée. Cette approximation peut utiliser l'information du gradient (comme les deux nouveaux algorithmes hybrides que nous avons développés) ou non (comme la SE avec Métamodèle de Emmerich et al. [*Emmerich et al. 2002*] ou les algorithmes proposé par Jin et al. [*Jin et al. 2000*] , [*Jin et al. 2001*], etc.). Avec cette approche, au sein d'une génération de l'AE, on peut avoir une partie des individus évalués avec la fonction objectif et l'autre partie avec la fonction approchée, ou bien toute la population évaluée par la fonction approchée.

Cette deuxième approche d'hybridation sera présentée par la description des deux nouveaux algorithmes hybrides que nous avons développés dans le chapitre II et celle de la SE avec Métamodèle (SE-Meta) dans le prochain paragraphe.

I.5. ALGORITHMES D'OPTIMISATION EMPLOYES

Nous avons testés 5 différents algorithmes pour résoudre nos problèmes d'optimisation de forgeage : BFGS, SCPIP, la SE-Meta et les deux algorithmes d'optimisation que nous avons développé. Parmi les 5 algorithmes utilisés, nous ne revenons pas sur l'algorithme BFGS car il est tout à faire classique. *Les deux nouveaux algorithmes hybrides, qui sont notre contribution originale aux méthodes d'optimisation, font l'objet du chapitre qui suit*. Cette section présentera les deux algorithmes SCPIP et la SE-Meta.

I.5.1. Algorithme SCPIP

Nous ne présentons pas les détails du SCPIP car il est assez complexe, nous présentons ici seulement son principe général de fonctionnement. L'algorithme complet est détaillé dans [*Zillober 2002*].

L'algorithme SCPIP [*Zillober 2002*] est un algorithme de type quasi Newton. Il est une combinaison de la méthode d'approximation convexe SCP (Sequential Convex Programming) avec la méthode de résolution par point intérieur (IP – Interior Point) pour chercher l'optimum global du problème (lorsque le problème est multi optima). Il a besoin du gradient de la fonction coût pour fonctionner. Construit sur la base de la méthode des asymptotes mobiles [*Svanberg 1987*], la méthode SCP remplace le problème d'optimisation original, difficile, par une série de sous-problèmes distincts, convexes et plus aisés à résoudre, et qui sont construits au cours des itérations :

- La fonction objectif et les contraintes d'inégalités sont remplacées par des approximations convexes,

- Les contraintes d'égalités sont linéarisées.

Ces sous-problèmes sont ensuite résolus par l'application de la méthode de point intérieur, qui est présentée dans [*Zillober 2001*].

Cet algorithme est considéré comme une méthode robuste pour résoudre des problèmes avec contraintes et fortement non-linéaires.

I.5.2. Stratégie d'évolution avec Méta-modèle

La Stratégie d'Evolution avec Métamodèle (SE-Métamodèle) a été développée à l'université de Dortmund et nous a été proposé dans le cadre du projet COST 526 APOMAT. Elle est une stratégie d'évolution utilisant une fonction d'approximation (métamodèle) dans le but de diminuer le nombre d'évaluations exactes de la fonction coût. Elle utilise donc l'organigramme d'un algorithme évolutionnaire canonique avec deux différences mises en relief sur la *Figure 1.9* :

- au lieu de commencer avec une population initiale normale, il utilise la méthode d'échantillonnage aléatoire pour créer une base de données initiale qui contient *2*n* (n est le nombre de paramètres à optimiser) de points initiaux. Ces points initiaux donc sont choisis dans tout l'espace de recherche.

- il évalue exactement seulement *20%* la population d'enfants à chaque génération et enrichit la base de données en y ajoutant ces individus évalués.

Nous allons donc maintenant expliquer le métamodèle, ce qui est le point clé de cette méthode d'optimisation.

Le Méta-modèle utilisé est une forme de surface de réponse qui permet d'approcher les valeurs de la fonction coût aux points où elle n'a pas encore été évaluée, à partir des valeurs déjà calculées. Ce modèle contient également une estimation de l'erreur d'approximation. Il permet ainsi d'éviter l'évaluation exacte de 80% des individus, a priori les moins performants d'une population, en tenant compte cette erreur d'approximation.

Le métamodèle présenté dans cette section est celui basée sur la méthode de Krigeage, comme méthode d'interpolation spatiale. Le Krigeage porte le nom de son fondateur, l'ingénieur minier D.G. Krige [*Krige 1951*], qui a développé des méthodes statistiques empiriques afin de déterminer la distribution spatiale de minerai à partir d'un ensemble de forage. C'est cependant le français Matheron [*Matheron 1963*] qui a formalisé l'approche en utilisant les corrélations entre forages pour la répartition spatiale. Dans les années récentes, le Krigeage était utilisé en géostatistiques [*Wackernagel 1998*] et en optimisation [*El-Bektagy et al. 1999*], [*Ralte 1998*], [*Trosset et al. 1997*].

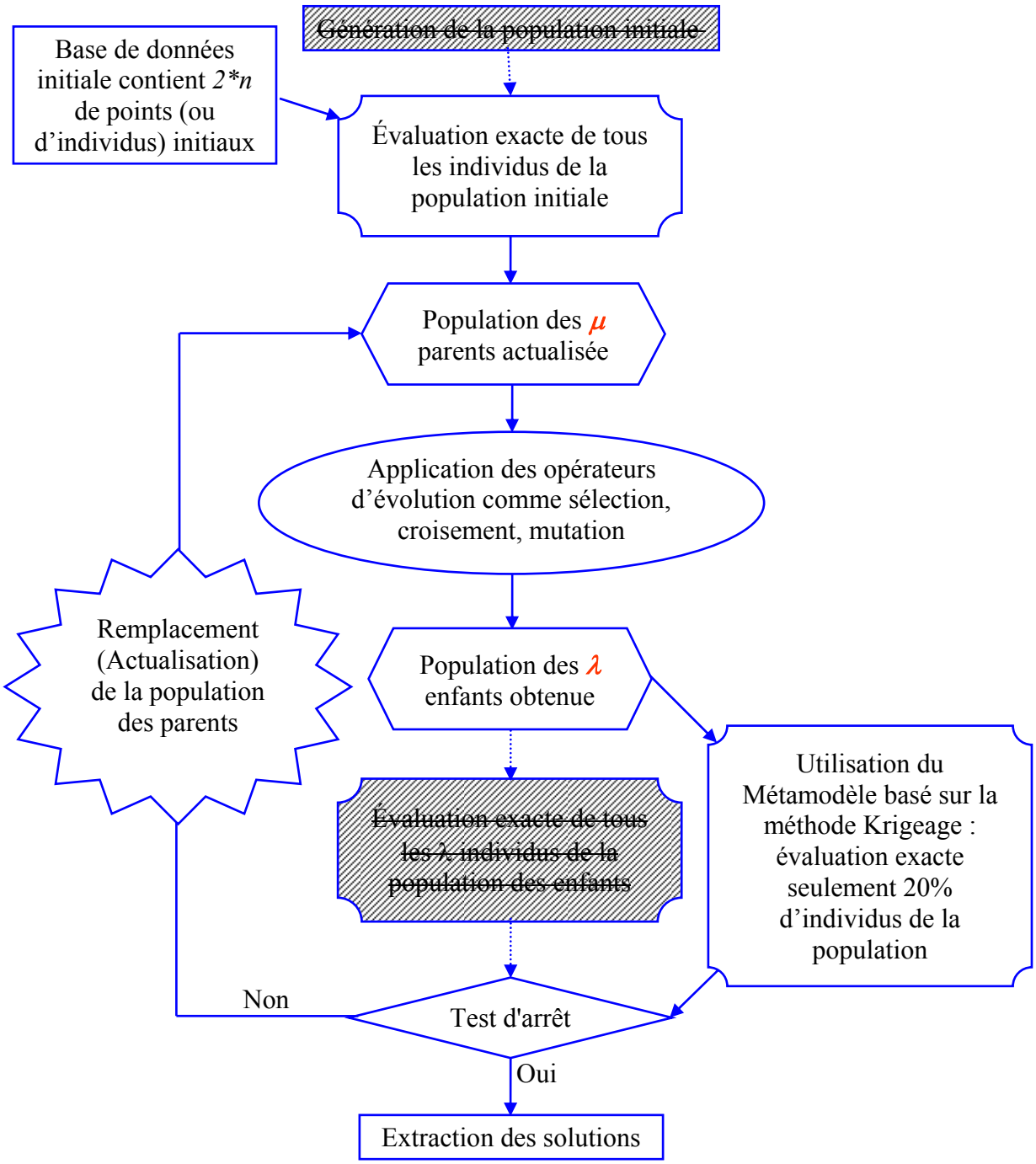

Figure 1.9. Organigramme de la SE-Méta basée sur la méthode Krigeage

Supposons que nous ayons une base de données composée de plusieurs points $x_1,...,x_m \in \Re^n$ déjà évalués et que $y = [y_1,...,y_m] = [f(x_1),...,f(x_m)]$ sont les résultats des évaluations exactes effectuées à chaque point.

En utilisant la méthode Krigeage, on approxime la valeur de la fonction objectif $f(x)$ d'un point x de l'espace de recherche par la fonction approchée $\hat{f}(x)$. Cette fonction $\hat{f}(x)$ est une réalisation Gaussienne stochastique isotrope de moyenne inconnue β et de covariance de forme :

$$c(u,v) = \sigma^2 r_\theta(u, v) \tag{1.23}$$

où $r_\theta(u, v)$ est la fonction de corrélation Gaussienne définie par $r_\theta(u, v) = e^{-\theta\|u-v\|^2}$ avec θ le paramètre gaussien inconnu, $u, v \in \Re^n$ et σ^2 la variance inconnue.

La construction de la fonction approchée à base du krigeage revient maintenant à l'estimation des trois inconnus β, σ^2 et θ [*Santner et al. 2003*]. L'idée de base de leur estimation est de minimiser l'erreur quadratique de prédiction suivante :

$$MSPE = E\left[\hat{f}(x) - f(x)\right]^2 \tag{1.24}$$

Cette idée ressemble à celle de minimiser l'erreur au sens des moindres carrés dans une méthode de surface de réponse. Il est courant d'utiliser la méthode de "*l'Estimation de la Probabilité Maximale (EPM)*" (Maximum Likelihood Estimation) pour estimer les inconnues β, σ^2 et θ. Cette méthode repose sur la même hypothèse que celle de Krigeage, c'est-à-dire que les mesures expérimentales (les points déjà évalués) résultent d'un processus Gaussien stochastique. La maximisation de la fonction de probabilité par rapport aux inconnues β, σ^2 et θ donne la valeur de l'approximation $\hat{f}(x)$ en tout point *x,* qui est le "Meilleur Prédicteur Linéaire Objectif" (Best Linear Unbiased Predictor – BLUP) de la réponse $f(x)$ [*Martin et al. 2003*] [*Lophaven et al. 2002*] :

$$\hat{f}(x) = \hat{\beta} + \left(y - I\hat{\beta}\right)^T R\left(\hat{\theta}\right)^{-1} r\left(x;\hat{\theta}\right) \tag{1.25}$$

où $R\left(\hat{\theta}\right)$ est la matrice symétrique $m \times m$ de composante $R_{ij} = r_{\hat{\theta}}(x_i, x_j)$ et $r\left(x;\hat{\theta}\right)$ le vecteur dimension *m* de composantes $r_i = r_{\hat{\theta}}(x_i, x)$.

Il faut noter que β et $R(\theta)$ ne dépendent que de θ et sont remplacés par leurs estimateurs $\hat{\beta}$ et $R\left(\hat{\theta}\right)$ (on suppose θ connu). On peut démontrer que $\hat{\beta}$ est l'estimation généralisée au sens des moindres carrés (Generalized Least Squares estimate) du coefficient de régression β [*Lophaven et al. 2002*] [*Martin et al. 2003*] [*Santner et al. 2003*]. Il prend la forme suivante :

$$\hat{\beta} = \left[I^T R\left(\hat{\theta}\right) I\right]^{-1} I^T R\left(\hat{\theta}\right)^{-1} y \tag{1.26}$$

Similairement à l'estimation de β, on peut aussi utiliser la méthode EPM pour estimer la variance $\hat{\sigma}^2$, ce qui donne :

$$\hat{\sigma}^2\left(\hat{\theta}\right) = \frac{1}{m}\left[y - \hat{\beta}\right]^T R\left(\hat{\theta}\right)^{-1}\left[y - I\hat{\beta}\right] \tag{1.27}$$

Le paramètre gaussien $\hat{\theta}$ (valeur optimale de θ) peut maintenant être estimée par la méthode EPM [*Martin et al. 2003*] en minimisant la quantité suivante :

$$\Pi = m\,log\left[\hat{\sigma}^2\left(\hat{\theta}\right)\right] + log\left[det\, R\left(\hat{\theta}\right)\right] \tag{1.28}$$

Une fois $\hat{\theta}$ connu, $\hat{\beta}$ et $\hat{\sigma}^2$ peuvent être calculés facilement en utilisant les équations (1.26) et (1.27). Maintenant, le BLUP $\hat{f}(x)$ de $f(x)$ est donné par l'équation (1.25). La valeur estimée $\hat{f}(x)$ est assortie d'une erreur (au sens des moindres carrés) [*Martin et al. 2003*] qui est estimée par :

$$\hat{MSE}(x) = \sigma^2 - \sigma^2 \begin{bmatrix} I & r(x;\hat{\theta})^T \end{bmatrix} \begin{bmatrix} 0 & I \\ I & R(\hat{\theta}) \end{bmatrix} \begin{bmatrix} I \\ r(x;\hat{\theta}) \end{bmatrix} \tag{1.29}$$

Cette erreur s'annule au cas où *x* se confond à un point déjà évalué. La *Figure 1.10* présente un exemple d'approximation $\hat{y} = \hat{f}(x)$ par la méthode de Krigeage en une dimension (pour un seul paramètre à optimiser). Dans cet exemple, la base de données contient 3 points déjà évalués.

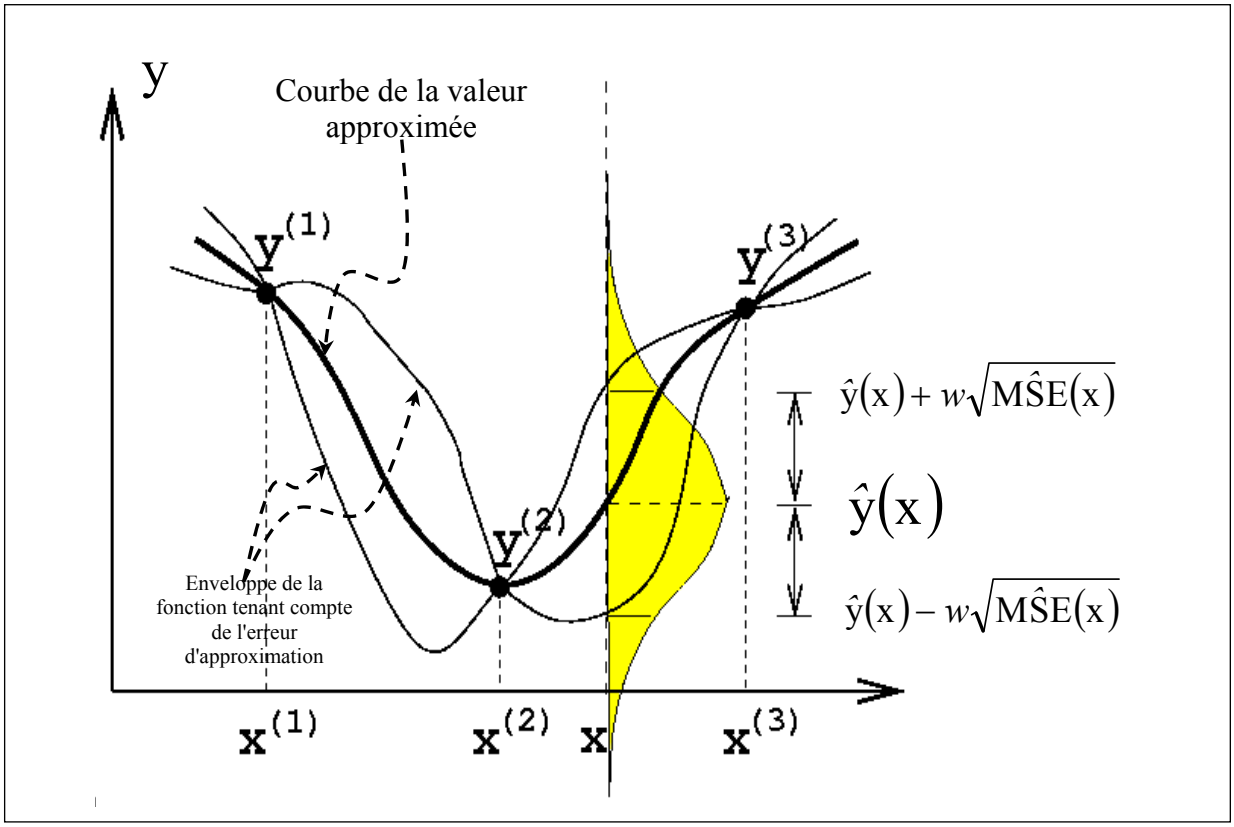

Figure 1.10. Approximation par la méthode de Krigeage

La précision de l'approximation dépend donc de la densité des points déjà évalués. Dans les zones avec une haute densité, l'approximation est plus précise et la valeur de l'erreur $\hat{MSE}$ est petite. Pour améliorer la qualité de l'approximation, de nouveaux points sont évalués et ajoutés à chaque génération à la base de données pour l'enrichir. La précision de l'approximation est alors augmentée, et les points choisis pour l'évaluation exacte ont plus de chance d'être les meilleurs individus de la génération.

Le choix des individus exactement évalués pour le méta modèle de krigeage est réalisé en utilisant l'expression (1.30). Ce critère est basé sur la valeur estimée de la fonction coût (1.25), corrigée par l'erreur d'approximation (1.29).

$$S_c(x) = \hat{f}(x) - w\sqrt{M\hat{S}E(x)} \qquad (1.30)$$

Le paramètre w ($w > 0$) a pour but d'équilibrer l'influence du terme d'erreur d'approximation. Si le problème d'optimisation est supposé raide, une grande valeur de w peut améliorer la capacité de localiser l'optimum global. Au contraire, une valeur basse de w va conduire l'algorithme vers un optimum local plus rapidement. La valeur de w = 1,0 est utilisée pour la SE-Meta dans cette étude.

Il existe une différence entre l'utilisation de l'approximation simple de la fonction coût (1.25) et celle basée sur le critère (1.30). Avec $\hat{f}(x)$, seuls les candidats les plus prometteurs sont choisis. Avec le critère (1.30) le choix est étendu pour explorer les zones d'améliorations potentielles pas encore explorées, en prenant compte l'erreur d'approximation, tout en examinant les individus ayant un potentiel de bonnes performances. La *Figure 1.11* présente un exemple, en une dimension, de la règle de sélection des individus ayant le meilleur potentiel avec le critère (1.30). Seuls les 20% des individus ayant la valeur de $S_c(x) = \hat{f}(x) - w\sqrt{M\hat{S}E(x)}$ la plus basse sont choisis.

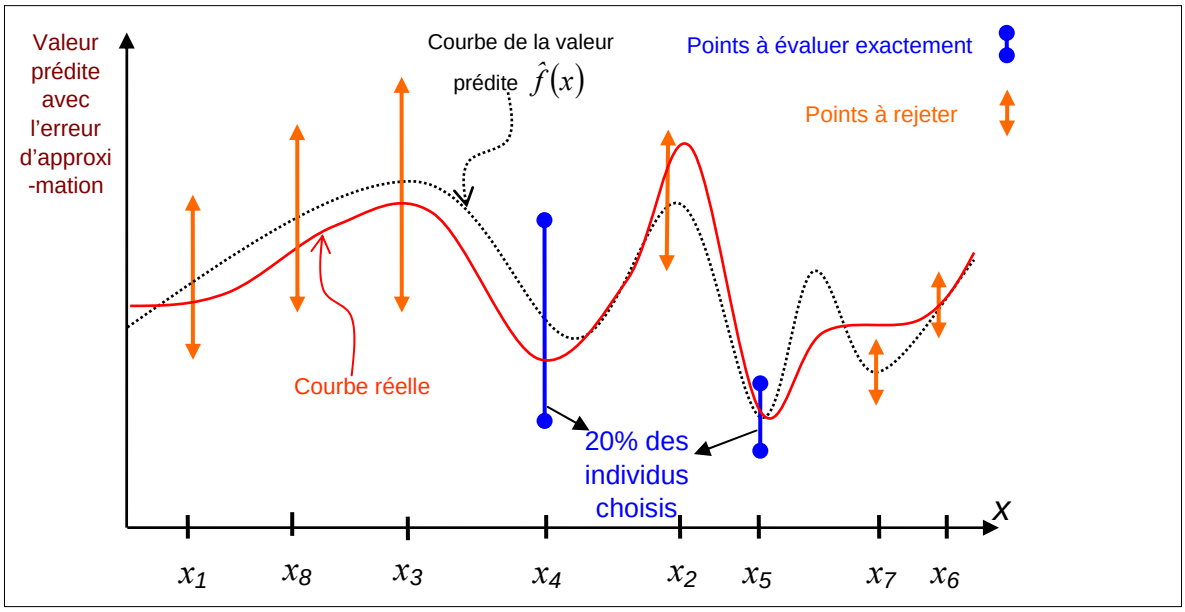

Figure 1.11. Règle de sélection des individus ayant le meilleur potentiel, utilisée par le Métamodèle

Le métamodèle de krigeage utilisé dans ce travail est basé sur les k voisins les plus proches de chaque point. La valeur de k varie entre 15 et 20. L'augmentation de k améliore légèrement l'approximation, mais aussi le temps de calcul des approximations.

Cette SE-Meta a été donc appliquée avec succès pour résoudre plusieurs problèmes d'optimisation difficiles comme l'optimisation d'ailes d'avions [*Emmerich et al 2002*]. Dans le

cadre de ce travail, nous l'utilisons pour résoudre des problèmes d'optimisation de forme du forgeage 3D.

I.6. CONCLUSION

Nous avons présenté dans ce chapitre une vue générale sur les différents algorithmes d'optimisation existants. Dans le chapitre qui suit, nous allons décrire les deux nouveaux algorithmes hybrides que nous avons développés pour l'optimisation globale.